CMPSC 465 Data Structures & Algorithms
Spring 2024 Mehrdad Mahdavi and David Koslicki

Worksheet 3

Wednesday, January 31, 2023

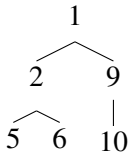
1. **Heap Sort.** Please consider the following array, which represents a min heap.

$$A = [1,2,9,5,6,10].$$

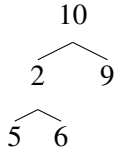
Suppose we remove the element at position 0 of the heap. How does the resulting heap look? Write both the array and tree representation of the heap.

Solution:

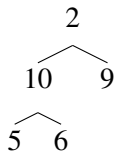
Our initial tree looks like this:



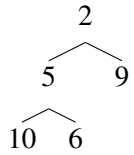
1. Firstly, we remove the node 1, and replace it with the last node i.e. 10.



2. Then we run Heapify-Down from the root. That is, we choose the smaller of the children, in this case between 2 and 9 is *2*. This element is now swapped with 10.



3. Similarly, we choose the smaller number between 5 and 6, then we swap to give us the result.



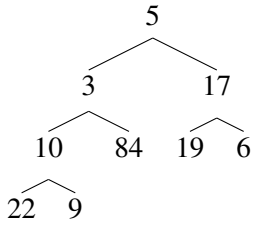
The array representation for this heap is $[2,5,9,10,6]$.

2. Heaps. How does the heap looks like after we run Build-heap function on the array

$$A = [5, 3, 17, 10, 84, 19, 6, 22, 9]$$

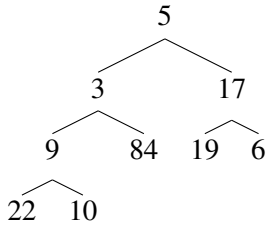
.

Our initial tree looks like this:

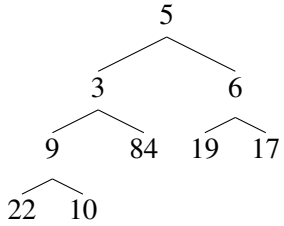


We run Heapify-Down on this tree, starting from node $\lfloor \frac{n}{2} \rfloor - 1$ down to 0 as follows:

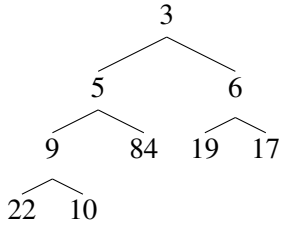
1. Firstly, we swap node 9 and 10.



2. Then we swap 17 and 6.



3. Then we swap 3 and 5.



We got the min-heap, so we do not need to do anything further.

3. **Heaps.** Show that an n-element binary heap has height $\lfloor \log_2 n \rfloor$.
Solution: A binary heap is by definition a full binary tree, meaning every level can only contain nodes if every above level is already full. Therefore, the number of nodes in a tree of height h is greater than or equal to 2^h. For example, A heap of height 2 must have at least 4 nodes (including 1 on level 2). We can similarly show that the number of nodes is less than or equal to $2^{h+1}-1$. This is because a heap of height h cannot have nodes on level $h+1$. For example, a heap of height 2 can have at most 7 nodes. Adding an eighth node would increase the height to 3. Therefore, $2^h \leq n \leq 2^{h+1}-1 < 2^{h+1}$. Taking the logarithm gives $h \leq \log_2 n < h+1$. Since h is an integer, $h = \lfloor \log_2 n \rfloor$.

4. **Creating heaps.** We can build a heap by repeatedly inserting the elements into the heap. Would this always create the same heap as Build-Heap when run on the same input array? Prove that they do, or provide a counterexample. **Solution:** Consider the array $[3,2,1]$. If we insert 3, 2 and 1 into the heap in that order, then the resulting heap array is $[1,3,2]$. But Build-Heap generates $[1,2,3]$. In general it is possible to make many different heaps from the same set of values.

5. **Find k-th smallest.** Given an array of n elements and an integer $k \geq 0$ we would like to find the k-th smallest element in the array.

    1. Provide an algorithm for this problem that uses a min heap with running $O(n + k \log n)$.
    2. Provide an algorithm for this problem that uses a max heap with running $O(n \log k)$.
    3. Which algorithm has better running time?

    Solution

    1. The idea is to construct min heap of size n by calling Build-Heap. Once the min heap is created we will delete the root of the heap $k-1$ times. After this, the root node would contain the k-th smallest element. The running time of Build-Heap is $O(n)$, and removing $k-1$ elements takes $O(k \log n)$. Hence, the running time is O(n + k log n).
    2. The idea is to create max heap and insert the first k elements of the array into the heap. Then, for each of the remaining elements of the array, if it is smaller than the value of the root node, we will delete the root of the heap and insert this new element. By the end of the process, we will only be left with a max heap containing the k smallest elements, and the root node will be the k-th smallest element. The running time will be $O(n \log k)$.
    3. The method using min heap can be more efficient for some values of k. When $k = O(1)$ or $k = \Theta(n)$, both methods have the same O running time. However, when $k \to \infty$ as $n \to \infty$ and k is not $\Theta(n)$, for example $k = \sqrt{n}$ or $k = \log n$, the method using min-heap is more efficient. Note that k cannot be greater than n.